B17

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 915 434 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
12.05.1999 Bulletin 1999/19

(51) Int. Cl.[6]: **G06T 15/50**

(21) Application number: **98108787.7**

(22) Date of filing: **14.05.1998**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **06.11.1997 US 965569**

(71) Applicant:
MITSUBISHI DENKI KABUSHIKI KAISHA
Tokyo 100-8310 (JP)

(72) Inventor: **Gibson, Sarah F.**
Arlington, MA 02174 (US)

(74) Representative:
Pfenning, Meinig & Partner
Mozartstrasse 17
80336 München (DE)

(54) System for depicting surfaces using volumetric distance maps

(57) A volumetric data representation for graphical objects encodes the distance of data elements to the nearest object surface, with means being provided for estimating the direction an magnitude of surface normals from the distance values. The subject system can be used to accurately reconstruct the normal direction of a slowly varying surface in volume sampled data for high quality Volume Rendering, accurate force reflection in haptic feedback, and accurate force calculation and object penetration estimation in physically-based graphics.

EP 0 915 434 A2

Description

FIELD OF THE INVENTION

[0001] This invention relates to Volume Graphics and more specifically to a discrete object representation that allows accurate reconstruction of surface normals for Volume Rendering, haptic exploration of volumetric models using force feedback, and physically-based calculations of interactions between graphical objects.

BACKGROUND OF THE INVENTION

[0002] In Volume Graphics, objects are represented as arrays of sampled data elements. Volumetric object representations have some advantages over surface-based graphical object representations because they can represent object interiors. This is important both for visualizing the internal structure of complex objects and for physically-based modeling of actions such as object deformation or the cutting and tearing of volumetric objects. Using volumetric object representations to model physically-based object interactions is discussed, for example, by S. Gibson in "Beyond volume rendering: visualization, haptic exploration, and physical modeling of voxel-based objects", in Proc. Eurographics workshop in Visualization in Scientific Computing, pp. 10-23, Chia, Italy, 1995. Using a volumetric object representation to model object deformation is discussed by S. Gibson in "3D ChainMail: a fast algorithm for deforming volumetric objects", in Proc. Symposium on Interactive 3D Graphics, Providence, RI, pp. 149-154, 1997.

[0003] Volume Rendering is technique used for displaying volumetric objects on a 2D computer monitor. There are several approaches to Volume Rendering, including ray casting where rays from each pixel in the image plane are cast into the object, accumulating color as they pass through the volumetric object; and splatting where each element in the object is projected onto the image plane, contributing color to the rendered image. For Volume Rendering, information about the color, transparency, surface normal, and material reflectance properties for each volume element are needed. These are often derived from two values stored for each element: a measured sample intensity or density value, and a material classification. For example, the density value might be used to calculate the element's opacity and surface normal, while the classification might be used to assign colors and reflectance properties to object elements.

[0004] In computer graphics, the quality of a rendered image is greatly improved by adding an illumination model to the rendering process. An illumination model allows the calculation of lighting effects such as spot lighting from specific light sources, reflections from object surfaces, and shadows. In computer graphics, a typical illumination model is the Phong illumination model, which expresses the light reflected from an object surface as follows:

$$I_r = K_a I_a I_{obj} + K_d I_L I_{obj}(NL) + K_s I_L I_{obj}(RV)^n,$$

where I_r, I_a, I_{obj}, and I_L are the reflected, ambient, object, and light source colors, K_a, K_d, and K_s are the ambient, diffuse, and specular reflection coefficients, N is the surface normal of the object, L is the direction of the light source, V is the viewing direction, and R is a function of N and L

[0005] In the Phong illumination model, an accurate representation of the object surface normal is essential. Inaccurate surface normals can have a large effect on the perceived shape and texture of a graphical object. Since, in most graphical representations, the surface normal is known accurately at only a relatively small number of positions, estimating the surface normal at arbitrary object positions is very important for high quality rendering. In Volume Graphics, estimating surface normals is especially challenging because accurate surface normals are not usually stored in the data. Instead, surface normals are usually estimated from data intensity or density values.

[0006] It is important to note that the calculation of accurate surface normals in Volume Graphics is not only needed for Volume Rendering. An accurate surface normal estimation is essential for haptic data exploration where a force feedback device is used by the user to tactually explore the surface of a graphical object. Accurate surface normal estimation is essential in this application because the sense of touch is even more sensitive than the eye to surface irregularities. An accurate surface normal is also important in modeling physically-based interactions between objects because the calculation of impact and frictional forces between objects depends on these normals.

[0007] Many methods have been used to add lighting effects, or shading, to Volume Rendered images. These methods depend on the data representation. Different methods are used when the data is represented as a binary field where data samples are either empty or full of a given material, or as a sampled continuous field, such as a sampled intensity or material density field. Many methods for shading volumetric objects are discussed, for example, in "Volume Visualization", edited by Arie Kaufman, and published by the IEEE Computer Society Press, Los Alamos, CA, in 1991. The more sophisticated of these methods require surface normal estimates at sampled data points.

[0008] In order to estimate surface normals when the object is represented as sampled continuous field, Volume Rendering often takes advantage of the fact that the image intensity or material density values change fastest at object surfaces. The directional change in image intensity can be represented by the 3D image gradient vector, which, at any

image position, points in the direction of fastest change in intensity values and which has a magnitude proportional to surface strength. Hence, this gradient vector is often used in Volume Rendering to estimate surface normals for illumination.

[0009] There are many ways to estimate image gradient vectors from sampled data. One straight-forward method is called the central difference method. Here the gradient vector, (g_x, g_y, g_z) is estimated from the 3D image intensity I (x, y, z) by:

$$g_x = I(x+1,y,z) - I(x-1,y,z),$$

$$g_y = I(x,y+1,z) - I(x,y-1,z),$$

$$g_z = I(x,y,z+1) - I(x,y,z-1).$$

[0010] Other image gradient operators include Sobel operators, cosinc filters, and more complex filters such as those discussed by S. Marschener and R. Lobb in "An evaluation of reconstruction filters for Volume Rendering", Proc. IEEE Visualization '94, pp. 100-107, Washington, DC, 1994, by T. Moller, R. Machiraju, K. Mueller, and R. Yagel in "Evaluation and design of filters using a taylor series expansion", IEEE Transactions on Visualization and Computer Graphics, Vol. 3, No. 2, pp. 184-199, 1997, and by M. Bentum, B. Lichtenbelt, and T. Malzbender in "Frequency analysis of gradient estimators in volume rendering", IEEE Transactions on Visualization and Computer Graphics, Vol. 2, No. 3, pp. 242-254, 1996.

[0011] S. Wang and A. Kaufman calculate smooth surface normals from volumetric data in "Volume-sampled 3D modeling", in IEEE Computer Graphics and Applications, pp. 26-32, Sept. 1994. They start with binary segmented data and the apply a Gaussian filter to the object intensity or density values. They then estimate surface normals as the gradient of this smoothed image.

SUMMARY OF THE INVENTION

[0012] In contradistinction to the above methods, in the subject invention, a volumetric distance map in which distances to the nearest object surface are stored for each volume element or voxel, is used to improve the surface normal estimation for volumetric objects. After having created a discrete volumetric distance map, one can calculate a gradient vector of the distance field for any point in the volume. Because the gradient vector derived from this distance map points in the direction of the surface, it provides the surface normal direction. The distance itself can be used to derive the surface normal magnitude.

[0013] While others have used volumetric distance maps in Volume Rendering, they have been used exclusively for speeding up the ray-casting rendering algorithm. The distance value are used in these systems as minimum distances to the next non-empty volume elements to adjust the step size along rays as they are cast into the subject. Two research papers that use a volumetric distance map to accelerate Volume Rendering are "Acceleration of ray-casting using 3D distance transforms", by K. Zuiderveld, A. Koning and M. Viergever in Proc. Visualization in Biomedical Computing, Chapel Hill, NC, pp. 324-335, 1992 and "Fast surface rendering from raster data by voxel transversal using chessboard distance", by M. Sramek in Proc. IEEE Visualization '94, pp. 188-195, Washington, DC, 1994.

[0014] Volumetric object representations have some advantages over surface-based graphical representations because they can incorporate complex interior structure. This is important both for visualizing internal structure and for modeling the physical behavior of 3D objects. However, in addition to interior structure, an accurate representation of surface is needed for high quality rendering, for haptic modeling, and for physics-based graphics. The subject invention addresses this need for an accurate representation of surfaces within volumetric data.

[0015] One reason that it is difficult to reconstruct surfaces from volumetric intensity or density values has to do with the discrete nature of the data. Information theory tells us that in order to reconstruct a signal exactly from sampled data, the signal must be sampled at twice the highest frequency component in the signal. Since a surface represents a sudden change in the image intensity or density value, the image signal at a surface has infinite frequency components and hence no sampling rate would be high enough for exact object reconstruction. Reconstruction errors are intensified with gradient estimation and these errors can lead to poor surface normal estimates and significant rendering artifacts.

[0016] The Gaussian filtering method of Wang and Kaufman prefilters the image data to reduce the high frequency components in a data volume and hence reduce aliasing artifacts in the gradient estimation. While this approach produces more pleasing images than those subject to aliasing, it also results in overly smoothed and rounded surfaces.

[0017] The subject invention encodes, into the volumetric data, knowledge of the presence, position and structure of known surfaces. Instead of storing object intensity or density values whose values usually change suddenly and drastically across a surface, the subject invention stores distances to the nearest surface element in each volumetric element. In one embodiment, distances are positive inside the object and negative outside of the object.

[0018] Note that:

1) the gradient vector of this distance map points in the direction of the surface normal;
2) a zero distance map value indicates the presence of a surface; and
3) a positive distance value indicates that the sample is inside the object. The distance to the nearest surface point varies linearly in the direction normal to a surface, and varies slowly in the direction parallel to the surface as long as the surface is relatively smooth, Hence, lower sampling rates and less complex gradient reconstruction filters can be used for accurate normal estimation with a distance map representation than with a sampled density representation.

[0019] It will be appreciated that in addition to using the distance map representation for calculating surface normals in Volume Rendering, distance maps can also be used to calculate surface normals for haptic exploration with a force feedback device and for calculating object penetration distances in interaction force calculations for physically-based modeling.

[0020] As there are many sources of data for a volumetric object representation, there are many ways to calculate the volumetric distance map. If the source of the data is an implicit or parametric representation of the surface, such as the equation of a sphere or a spline patch, then either geometric reasoning, or a constrained optimization technique can be used to find the minimum distance from any point in the volume to the defined surface. For a polygonal surface model, various algorithms used in graphical collision detection can be used to calculate the distance from any point in volume to the nearest surface polygon. For a discretely sampled continuous field, such as a density field for which no further knowledge of the surface is known, one method to generate distance measures would be to construct a polygonal spline patch surface model at iso-surface values using a technique such as Marching Cubes, described by W. Lorensen and H. Cline, in "Marching Cubes: a high resolution 3D construction algorithm", in Proc. SIGGRAPH '87, pp. 163-169, 1987, and then calculate distance from this constructed surface. For a binary segmented object, the binary object could be smoothed to form a continuous sampled density field and then treated as above, or distance could be calculated from the binary segmented data using measurements such as the city-block distance or Euclidean distance approximations that are described, for example, in "The image processing handbook", by J. Russ, published by CRC Press, in Boca Raton, FL, 1992. The resultant distance maps could also be filtered to obtain a smoother distance field.

[0021] The subject system finds application in computer graphics for generating lighting effects for volume rendering, for calculating reflected forces for volumetric objects in haptic rendering systems and for calculating physically-based response to interactions between volumetric graphical objects.

[0022] With respect to generating light effects in volume rendering, the calculation of the magnitude and direction of light reflected from an object's surface requires an estimate of the object surface normal. The subject system for calculating surface normals provides a means for generation these light effects for discrete volumetric objects.

[0023] With respect to calculating reflected forces in haptic rendering systems, the direction and magnitude of the force that is reflected in haptic rendering through use of a force feedback device depends on the normal of the graphical object's surface at the position of the end effector of the force feedback device and the distance of the end effector from the object surface. The reflected forces allow one to haptically explore, or "fee", the surface of the graphical object. The subject system of encoding distance to surface in a volumetric distance map and for calculating surface normals provides a means for generating the required reflected force direction and magnitude.

[0024] With respect to calculating physically-based responses to interactions between graphical objects, an interaction is detected when two objects penetrate each other in a physically-based simulation, and the forces caused by the interaction can be calculated from the depth of penetration of the two objects and the object surface normals at the contact point. The subject system of encoding distances to surfaces in a volumetric distance map and for calculating surface normals provides a means for detecting object penetration and for generating the required interaction force direction and magnitude. Object penetration of one object into a second object is detected when a point of the first object maps to a position in the distance map of the sec object that has a distance value which indicates that the point is inside the second object. The depth of penetration of the first object point into the second object can be calculated from the distance map of the second object. The depth of penetration can be combined with object surface normals calculated from the distance maps to calculate the direction and magnitude of interaction forces.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] These and other features of the Subject Invention will be better understood in relation to the Detailed Description in conjunction with the Drawings, of which:

Figure 1 is a diagrammatic representation of a computer system used for high quality rendering;

Figure 2 is a diagrammatic representation of vector components in the Phong illumination model;

Figure 3A is a diagrammatic representation of the vector components used for calculating the lighting effects in a polygonal graphical object model;

Figure 3B is a diagrammatic representation of a means for estimating the surface normal at a point, P, on a polygonal object model;

Figure 4 is a flow chart of Volume Rendering using ray-casting with an illumination model;

Figure 5A is a diagrammatic representation of Volume Rendering using ray-casting;

Figure 5B is a diagrammatic representation of the vector components needed at a ray sample point, P, for calculating the illumination effects;

Figure 6A is a typical data structure written in the C programming language for representing a volumetric element for Volume Rendering;

Figure 6B is a data structure written in the C programming language that encodes both the element density and the distance map;

Figure 7 is a diagrammatic representation illustrating the differences between the prior art and the subject invention for shading in Volume Rendering, with (a) representing the two dimensional graphical object, (b) representing the object's distance map, (c) representing the sampled version of (a), (d) representing the sampled version of (b), (e) depicting the gradient of (c), calculated using the central difference algorithm, and (f) depicting the gradient of (d), calculated using the central difference algorithm; and

Figure 8 is a block diagram of a system for generating distance maps for graphical objects from a variety of data sources.

DETAILED DESCRIPTION

[0026] Referring now to Figure 1, high quality rendering of a graphical object 2, is performed on a computer 4, with user input from the keyboard 6, and an input device 8, and displayed on a computer monitor 10.

[0027] Referring to Figure 2, in computer graphics, high quality image rendering usually incorporates an illumination model such as the Phong illumination model which estimates the color to the light reflected at a point P from a graphical object as a function of the color of the object, the color of the light source, the ambient color in the room, the reflection coefficients of the object, the direction, L, of the light source, the direction, V, as shown, of the eye 22, and the surface normal, N, at P. For a given light source and a fixed viewing direction, only the object color and the surface normal vary over the surface of the object. Accurate estimation of surface normals is essential for realistic rendering.

[0028] More particularly, it will be appreciated that the intensity of the light at each pixel of this image plane includes a component due to reflectance from the object surface. The refelectance intensity at point P1 is affected by the viewing direction V_1, from the image plane 20 to the object 22; the lighting direction, L_1, from the light source 24 to the point where the viewing ray intersects the object; and the surface normal, N_1, at that point.

[0029] In one embodiment, with the object being a cup, that which is necessary to accurately render the cup is to be able to obtain the intensity, light direction and color of a point on the cup. As to image plane 20, if one can establish a normal to point P_1 then one can describe the magnitude or intensity of light from that point as it relates to the viewing direction and the lighting direction. Since the voxel at point P_1 contains information as to color and intensity, all that is necessary to establish is the normal at point P_1 to properly establish the intensity and color of a corresponding point on the image plane.

[0030] Referring to Figure 3A, in polygon modeling, graphical objects are represented by surface polygons, e.g. 24, 26 and 28. When rendering polygonal models, surface illumination at a point P, 30, is calculated from the viewing direction, V, 32, the lighting direction, L, 34, and an estimate, N, 36, of the surface normal at the point P, 30, using an illumination model such as the Phong illumination model. Referring to Figure 3B, typically, accurate surface normals for the object are stored in the object model at the polygon vertices, 40, 42, 44, 46 and 48. Typically, the surface normal, N, 50, at the sample point P, 38, is estimated from the stored surface normals at the vertices of polygon 52 on which the sample point lies.

[0031] Referring to Figure 4, in ray casting for Volume Rendering, rays are cast into the volumetric data from each

pixel in the view plane 54. Each ray is sampled at equally spaced intervals as it traverses the volume 56. At each sample point, the object color 58, transparency 60, and surface normal 62, are estimated. These are used along with an illumination model to calculate the light reflected at each sample point 64. Once the color of each sample point is known, it is composited onto the ray color 66.

[0032] Referring to Figure 5A, rays from the view plane, e.g. 68, 70 and 72, are cast into the volume in the viewing direction, sampling the color of the volumetric data at equally spaced intervals, e.g. 74 and 76 and compositing the sample color onto the ray color. Referring to Figure 5B, in high quality Volume Rendering which uses an illumination model, the color of each element includes a component due to light reflected by the sample element. The color of the reflected light is calculated using an illumination model such as the Phong illumination model, and the resultant light color at the sample point, P, 78, depends on the direction, L, 80, of the light source, the direction, V, 82, of the viewing direction, and the surface normal, N, 84 at P, 78. Unlike polygonal models, accurate surface normals are not usually stored in the volumetric data. Hence, the surface normal, N, 78, at the sample point must be calculated from the density values stored in data elements near the sample point. Referring to Figure 6A, typically, in Volume Rendering, each element in the volume has a structure such as the one shown here, written in the C programming language. Each element contains a value 86 for the object density or intensity. Referring to Figure 6B, in the subject invention, the data structure for each element has an additional field 88 so that a value for the distance to the nearest surface can be stored for each element. In the subject invention, this distance is used to detect the presence of a surface and to estimate its normal direction. In one embodiment, the distance is positive inside of the object, negative outside of the object and zero on the object surface. The gradient of this distance field points in the direction of the surface normal. Surface are detected where the magnitude of the distance is below a small threshold.

[0033] Referring to Figure 7, differences between the prior art for estimating surface normals in Volume Rendering and the method of the subject invention are illustrated in 2D. 7(a) is a 2D object. The object is present where the image is black as illustrated at 90, and absent where it is white, as illustrated at 92. Figure 7(b) is the continuous distance map that corresponds to the object in Figure 7(a). Note that it has a value of 0.5 on the surface, or edge, of the 2D object 94; values that increase with distance from the surface and ranging from 0.5 to 1 inside the object; and values that decrease with distance from the surface and ranging from 0.5 to 0 outside of the object. Figure 7(c) is the sampled version of Figure 7(a), where sample points are black if they fall inside the object, e.g. 96, and white if the fall outside of the object, e.g. 98. Figure 7(d) is the sampled version of Figure 7(b), where the value of each sample point is the color of the distance map at the same position. Notice that the intensity values of the sample points in Figure 7(d) vary smoothly across the image. Because of this smooth variation, values of the distance field that lie between sample points can be reconstructed more accurately and with a lower order interpolation filter than values of the sampled image density field of Figure 7(c) can be reconstructed. Figure 7(e) is a sampled estimate of the gradient field of Figure 7(a) that was calculated from the sampled image of Figure 7(c) using a central difference gradient estimator. Note that the direction of the sample gradients can vary significantly for neighboring samples, e.g. 98 and 100, or 102 and 104. Using this gradient estimate in Volume Rendering would cause significant artifacts in the image. Figure 7(f) is an estimate of the surface normal directions of Figure 7(a) calculated using a central difference gradient estimator from the sampled image of Figure 7(d). The direction of the surface estimate is consistent throughout Figure 7(f) and it accurately represents the normal of the 2D object surface. The magnitude of the surface normal can be derived from the distance map in Figure 7(d). In this example, the surface normal magnitude is large when the distance map intensity is close to 0.5 and small otherwise.

[0034] Referring to Figure 8, there are different ways to calculate an object's distance map 106, depending on the object representation 108. If the object is represented implicitly or parametrically as at 110, such as the equation of a sphere or a spline patch, then either geometric reasoning or constrained optimization 112 can be used to find the minimum distance from any point in the volume to the defined surface. If the object is represented by a polygonal surface model 114, various algorithms used in graphical collision detection can be used to calculate the distance from any point in volume to the nearest surface polygon 116. For a discretely sampled continuous density field 118, one method to generate distance measures would be to construct a polygonal surface model at iso-surface values 120, and then calculate distances from this constructed surface 116. For a binary segmented object, the binary object could be smoothed as illustrated at 124 to form a continuous sampled density field and then treated as above as illustrated at 120 and 116, or Euclidean distances could be calculated from binary segmented data 126. The resultant distance maps could also be filtered to obtain a smoother distance field 128.

[0035] A program listing in C is presented hereinafter to illustrate the utilization of the subject system for use in generating lighting effects for volume rendering:

```
/* ..... volRender.c .....
 *
 * Basic volume rendering program.  Takes a data volume consisting of an
 * intensity and a distance-to-surface measure for each voxel and projects
 * the volume onto a 2D image.
 * Revised from earlier projection programs.
 *
 * Calculates object color, and transparency using the voxel intensity values
 * and shading based on a central difference of the distance values.
 * Compositing is done in front-to-back order and terminates at a max opacity.
 *
 * July, 1997 by S. Gibson,
 * MERL Cambridge.
 *
 */

/* includes */
#include "volRender.h"

/* Vis_Display global variables */
static int xSize, ySize, zSize;
static int imgSize;
static int nLights;
static int vertexOffset[8];

static float voxelSizeX, voxelSizeY, voxelSizeZ;
static float scaleX, scaleY, scaleZ;
static float R[3][3], T[3];
static float kd, ks, specularExp, visibility;
static float imageGain;

static VisLight lights[MAXLIGHTS];
static VisFVector view;

/*
 * ----------------------------------------------------------------------
 * main --
 *
 * ----------------------------------------------------------------------
 */

void main(int argc, char **argv)
{
  char dataFileName[60];
  char clutFileName[60];
  char tlutFileName[60];

  int i, j, k, index;
  int size, imgArea;
  int fd, n;

  XEvent event;

  VisVoxel *voxel;                                    /* 3D data volume */
  VisColor *charImage;                          /* projected image (char) */
  VisFColor *image;                  /* projected image (floating point) */
  VisColorLut clut;                     /* array of color lookup tables */
  VisTranspLut tlut;                 /* array of transparency lookup tables */

  /* initialize the rendering parameters */
  InitProjection(argv[1], dataFileName, clutFileName, tlutFileName);

  /* allocate space for the data and image arrays */
  size = xSize * ySize * zSize;
  voxel = (VisVoxel *) malloc(size * sizeof(VisVoxel));
```

```
    imgArea = imgSize * imgSize;
    image = (VisFColor *) malloc(imgArea * sizeof(VisFColor));
    charImage = (VisColor *) malloc(imgArea * sizeof(VisColor));

    /* open and read the data file */
    fd = open(dataFileName, O_RDONLY);
    read(fd, voxel, size * sizeof(VisVoxel));
    close(fd);

    /* clear the image to black */
    VisClearImage(image);

    /* render the volume data */
    VisProject(voxel, clut, tlut, image);

    /* scale the colors of the image to chars, correct for overflow */
    ColorsToChar(image, charImage, imgArea);

    /* write the image to a file */
    fd = creat("image.raw", PMODE);
    write(fd, charImage, imgSize * imgSize * sizeof(VisColor));
    close(fd);

    /* free the data allocation */
    free((VisVoxel *) voxel);
    free((VisFColor *) image);
    free((VisColor *) charImage);
}

/*
 * ----------------------------------------------------------------------
 * VisProject --
 *
 *   This procedure calculates the projected image by casing rays from the
 *   image plane throught the object, calculating the gradient, color,
 *   transparency, and shaded color contribution from each sample point
 *   along the ray and then compositing these values in a front-to-back
 *   ordering.
 * ----------------------------------------------------------------------
 */

void
VisProject(VisVoxel *voxel, VisColorLut clut, VisTranspLut tlut,
           VisFColor *image)
{
  int i, j, k, ii, jj, kk;
  int index, nIndex;
  int planeSize;
  int neighborIndex[8], values[8];

  float x, y, z;
  float dx, dy, dz, ddx, ddy, ddz;
  float temp0, temp1, temp2, temp3;
  float w[8];
  float transp, opacity, accTransp;
  float mag, sign, dist;

  VisFColor color, shadingColor;
  VisFVector grad;

  planeSize = xSize * ySize;

  /* 1) calculate contributions to the image plane by sampling along a ray
   * through the data in the kk direction in front-to-back order
   */
```

```
/*
 *  2) for each pixel (ii,jj) in the image plane, 1) find the
 *     corresponding coordinates (x,y,z) in the data volume for the kkth
 *     sample point along the ray, 2) trilinearly interpolate attributes
 *     of the sample point from the surrounding neighbors, 3) accumulate
 *     the color along the ray by compositing the color of the sample
 *     point onto the image plane pixel.
 */
index = 0;
for (jj = 0; jj < imgSize; jj++) {
  for (ii = 0; ii < imgSize; ii++) {
    accTransp = 1.0;
    kk = 0;
    color.r = color.g = color.b = 0.0;
    transp = 1.0;

    while ((kk < imgSize) && (accTransp > 0.001)) {
      /* 3) calculate the (x,y,z) coordinates of (ii,jj,kk) in image space */
      x = R[0][0] * ii + R[0][1] * jj + R[0][2] * kk + T[0];
      y = R[1][0] * ii + R[1][1] * jj + R[1][2] * kk + T[1];
      z = R[2][0] * ii + R[2][1] * jj + R[2][2] * kk + T[2];

      /*  make sure the (x,y,z) lies inside of the data volume */
      if ((x >= 0) && (x < xSize-1) &&
          (y >= 0) && (y < ySize-1) &&
          (z >= 0) && (z < zSize-1) ) {

        /* 4) calculate coordinate dependent weights and constants */
        i = (int) x; j = (int) y; k = (int) z;
        dx = x-i; dy = y-j; dz = z-k;
        ddx = 1-dx; ddy = 1-dy; ddz = 1-dz;

        temp0 = ddx*ddy; temp1 = ddx*dy; temp2 = dx*ddy; temp3 = dx*dy;

        /* weights for trilinear interpolation where the indices correspond
         * to the neighbors in the following way: 0->000, 1->001, 2->010,
         * 3->011, 4->100, 5->101, 6->110, and 7->111.
         */

        w[0] = temp0*ddz; w[1] = temp0*dz; w[2] = temp1*ddz; w[3] = temp1*dz;
        w[4] = temp2*ddz; w[5] = temp2*dz; w[6] = temp3*ddz; w[7] = temp3*dz;

        neighborIndex[0] = i + j*xSize + k*planeSize;
        dist = VisGetSampleDist(voxel, w, neighborIndex[0]);
        if (dist >= -0) {          /* add color only for interior elements */

          /* indices of 8 surrounding voxels where the indices correspond to
           * the neighbors in the following way: 0->000, 1->001, 2->010,
           * 3->011, 4->100, 5->101, 6->110, and 7->111.
           */
          for (nIndex = 1; nIndex < 8; nIndex++) {
            neighborIndex[nIndex] =
              neighborIndex[0] + vertexOffset[nIndex];
          }

          for (nIndex = 0; nIndex < 8; nIndex++) {
            values[nIndex] = (int) voxel[neighborIndex[nIndex]].value;
          }

          /*
           *  5) calculate the gradient, color, transparency, and shaded
           * color of the sample point from the surrounding neighbors.
           */
          VisGetSampleColor(values, w, clut, &color);
          transp = VisGetSampleTransp(values, w, tlut);
```

```
                if (nLights > 0) {

                  if (mag = VisGetSampleGrad(voxel, w; i, j, k, &grad) >= 0) {
                    /* find direction of grad relative to the view direction.
                     * If positive, this sample point contributes to the shading.
                     */
                    sign = VectorDot(grad, view);
                    if (sign >= 0) {
                      shadingColor = color;
                      VisGetShadedColor(grad, mag, &shadingColor);

                      color.r += shadingColor.r;
                      color.g += shadingColor.g;
                      color.b += shadingColor.b;
                    }
                  }
                }

                /* 6) project the voxel onto the view plane */
                image[index].r += accTransp * color.r;
                image[index].g += accTransp * color.g;
                image[index].b += accTransp * color.b;
                accTransp *= transp;
              }
            }
            kk++;
          }
          index++;
        }
      }

      return;
    }

    /*
     * ----------------------------------------------------------------------
     * VisGetSampleColor --
     *
     *   This procedure gets the sample color by trilinearly interpolating the
     *   colors of its 8 nearest neighbors after they have been determined
     *   using a color lookup table.
     *
     * ----------------------------------------------------------------------
     */

    void
      VisGetSampleColor(int *values, float *weight, VisColorLut clut,
                        VisFColor *color)
    {
      int i;
      VisColor temp;

      color->r = color->g = color->b = 0;

      for (i = 0; i < 8; i++) {
        temp = clut.color[values[i]];
        color->r += visibility * weight[i] * (float) temp.r / 255.0;
        color->g += visibility * weight[i] * (float) temp.g / 255.0;
        color->b += visibility * weight[i] * (float) temp.b / 255.0;
      }

      return;
    }
```

```
/*
 * ----------------------------------------------------------------------
 * VisGetSampleTransp --
 *
 *   This procedure gets the sample transparency by trilinearly interpolating
 *   the transparencies of its 8 nearest neighbors after they have been
 *   determined using a lookup table.
 *
 * ----------------------------------------------------------------------
 */

float
VisGetSampleTransp(int *values, float *weight, VisTranspLut tlut)
{
  int i;
  float transp = 0;

  for (i = 0; i < 8; i++) {
    transp +=  weight[i] * tlut.transp[values[i]];
  }

  transp /= 255.0;                                  /* max char value is 255 */
  return transp;
}


/*
 * ----------------------------------------------------------------------
 * VisGetSampleDist --
 *
 *   This procedure gets the sample distance by trilinearly interpolating
 *   the distances to nearest surfaces of its 8 nearest neighbors.
 *
 * ----------------------------------------------------------------------
 */

float
  VisGetSampleDist(VisVoxel *voxel, float *weight, int index)
{
  int i;
  float dist = 0;

  for (i = 0; i < 8; i++)
    dist +=  weight[i] * voxel[index + vertexOffset[i]].dist;

  return dist;
}


/*
 * ----------------------------------------------------------------------
 * VisGetSampleGrad --
 *
 *   This procedure gets the sample gradient from the distances of the
 *   neighboring elements using a central difference method.
 *
 * ----------------------------------------------------------------------
 */

float
VisGetSampleGrad(VisVoxel *voxel, float *weight,
                 int voxelI, int voxelJ, int voxelK, VisFVector *grad)
{
  int i, j, k, index;
  float mag;
```

```
    /* apply central difference operator to trilinearly interpolated distances
       of the voxel's 6 nearest neighbors to estmate the image gradient */
    /* note: samples must lie inside the data volume */
    if ((voxelI > 1) && (voxelI < xSize-2) &&
        (voxelJ > 1) && (voxelJ < ySize-2) &&
        (voxelK > 1) && (voxelK < zSize-2)) {
      index = voxelI + voxelJ*xSize + voxelK*xSize*ySize;

      grad->x = - VisGetSampleDist(voxel, weight, index + 1) +
        VisGetSampleDist(voxel, weight, index - 1);
      grad->y = - VisGetSampleDist(voxel, weight, index + xSize) +
        VisGetSampleDist(voxel, weight, index - xSize);
      grad->z = - VisGetSampleDist(voxel, weight, index + xSize*ySize) +
        VisGetSampleDist(voxel, weight, index - xSize*ySize);

      mag = sqrt(grad->x * grad->x + grad->y * grad->y + grad->z * grad->z);
    }

    else {
      mag = 0;
    }

    if (mag > 0) {
      grad->x /= mag;
      grad->y /= mag;
      grad->z /= mag;
    }

    else {
      mag = 0;
      grad->x = grad->y = grad->z = 0;
    }

    return mag;
  }

  /*
   * ----------------------------------------------------------------------
   * VisGetShadedColor --
   *
   *   This procedure calculates shaded color using a Phong shading algorithm
   *   with diffuse and specular reflection.
   *
   * ----------------------------------------------------------------------
   */

  void VisGetShadedColor(VisFVector grad, float mag, VisFColor *color)
  {
    int i;
    float dScale, temp, sScale;
    VisFColor shadeColor;

    shadeColor.r = shadeColor.g = shadeColor.b = 0;

    if (visibility != 0) {
      for (i = 0; i < nLights; i++) {
        /* get the diffuse lighting scaling factor */
        temp = VectorDot(grad, lights[i].lightVector);
        if (temp > 0)
          dScale = mag * kd * temp;
        else
          dScale = 0;

        /* get the specular light scaling factor.  Note the use of the halfway
```

```
            * vector assumes that the light and the viewing position are infinitely
            * far away (see Foley, Van Dam, et al.).
            */
           temp = VectorDot(grad, lights[i].hwVector);
           if (temp > 0)
             sScale = mag * ks * pow(temp, specularExp);
           else
             sScale = 0;

           shadeColor.r += visibility * lights[i].r * (dScale * color->r + sScale);
           shadeColor.g += visibility * lights[i].g * (dScale * color->g + sScale);
           shadeColor.b += visibility * lights[i].b * (dScale * color->b + sScale);
         }
       }

       *color = shadeColor;
       return;
     }
```

[0036] Having above indicated several embodiments of the Subject Invention, it will occur to those skilled in the art that modifications and alternatives can be practiced within the spirit of the invention. It is accordingly intended to define the scope of the invention only as indicated in the following claims.

Claims

1. In a volumetric data representation system for graphical objects in which graphical objects are generated from data elements,

   means for encoding the distance of a data element to the nearest object surface point; and,

   means responsive to the distance of said data element to the nearest object surface point for estimating a surface normal thereof.

2. The system of Claim 1, and further including means responsive to the distance of said data element to the nearest object surface point for estimating the magnitude of said surface normal.

3. The system of Claim 1, wherein said distance encoding means includes means for generating a discrete volumetric distance map and means for generating a gradient vector at any point in the space that contains said surface from said distance map, said gradient pointing in the direction of a surface normal.

4. The system of Claim 3, wherein a zero distance map value indicates the presence of a surface.

5. The system of Claim 3, wherein a positive distance value indicates that said data element is inside said object.

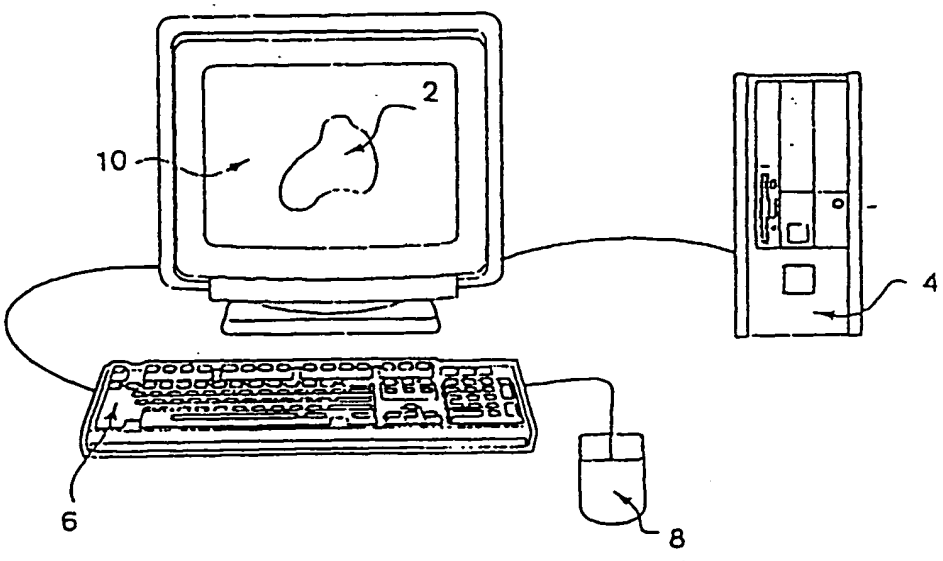

Fig. 1

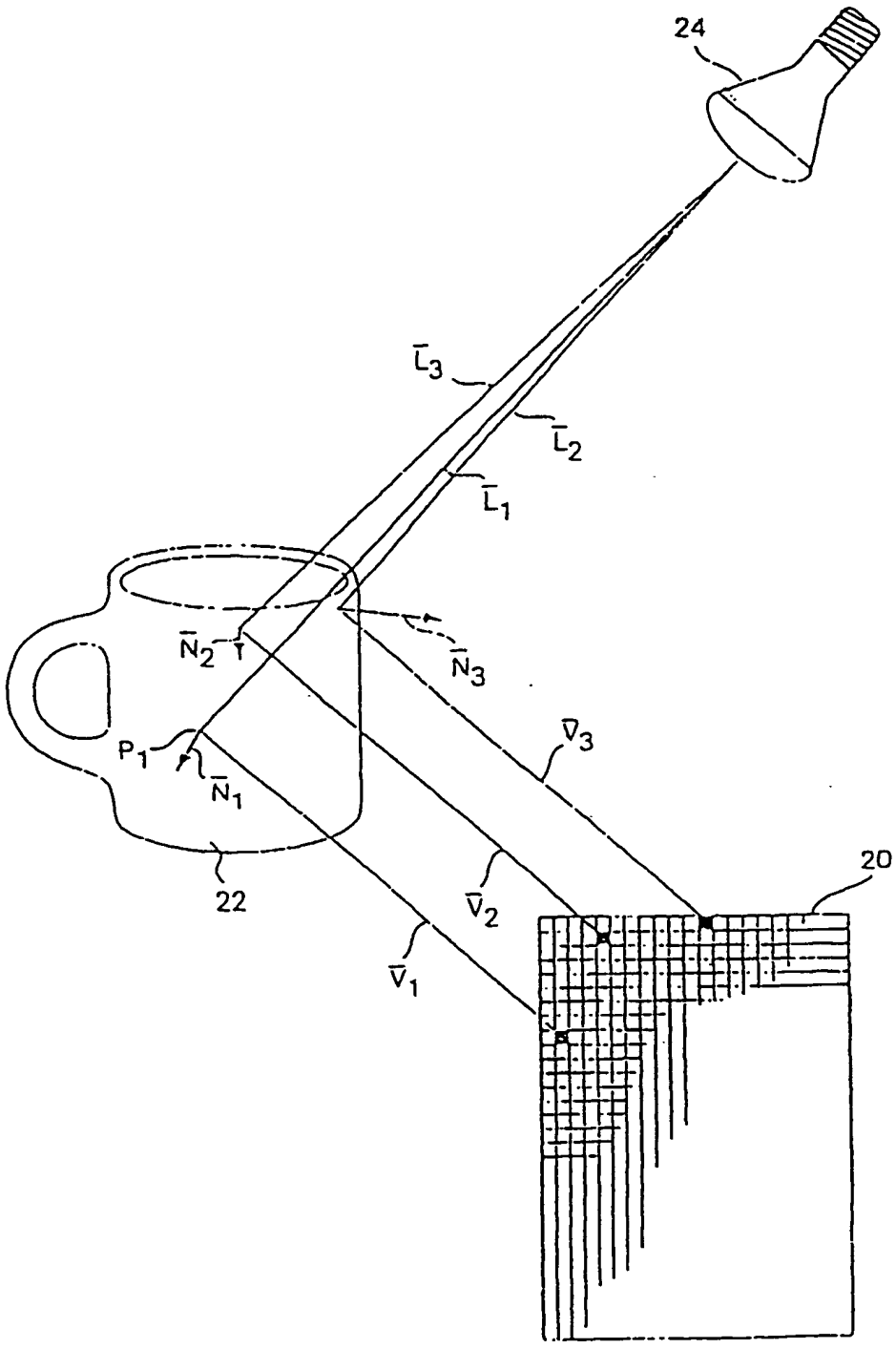
24
$\bar{L}_3$
$\bar{L}_2$
$\bar{L}_1$
$\bar{N}_2$
$\bar{N}_3$
P_1
$\bar{N}_1$
$\bar{V}_3$
$\bar{V}_2$
$\bar{V}_1$
22
20

Fig. 2

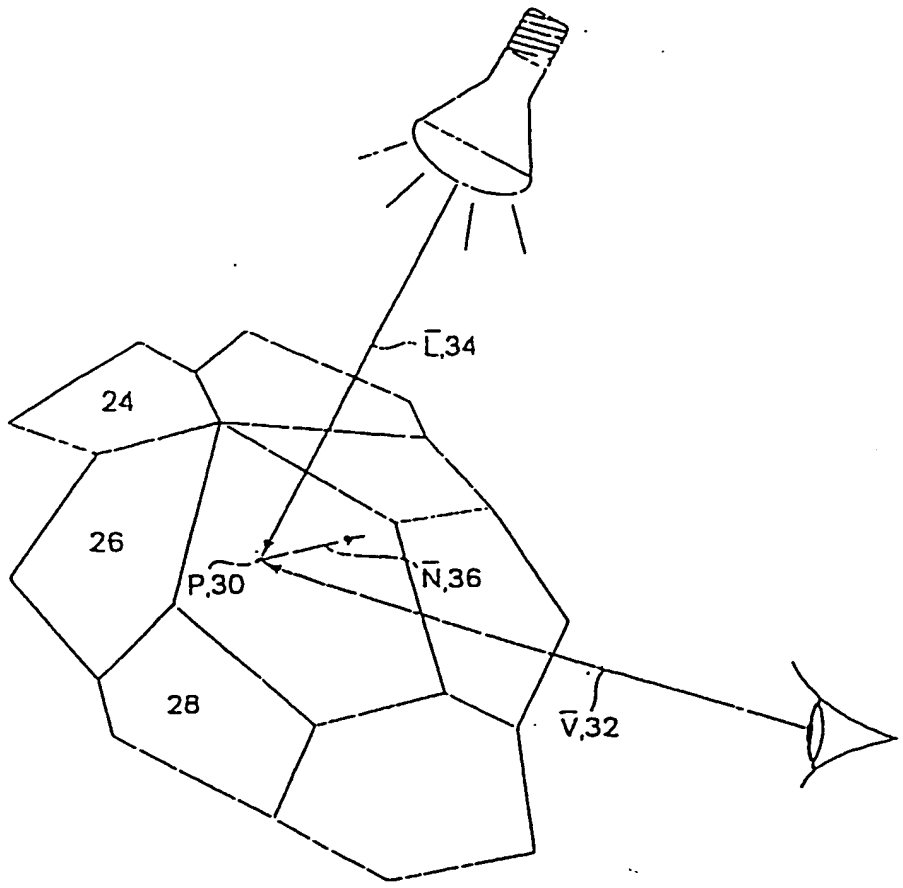

Fig. 3A

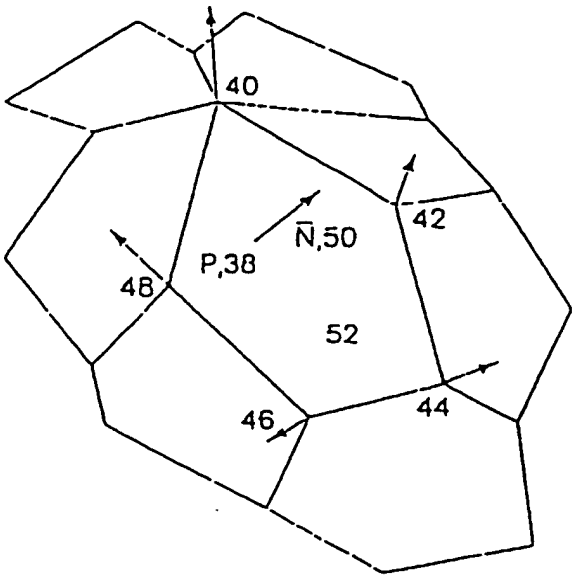

Fig. 3B

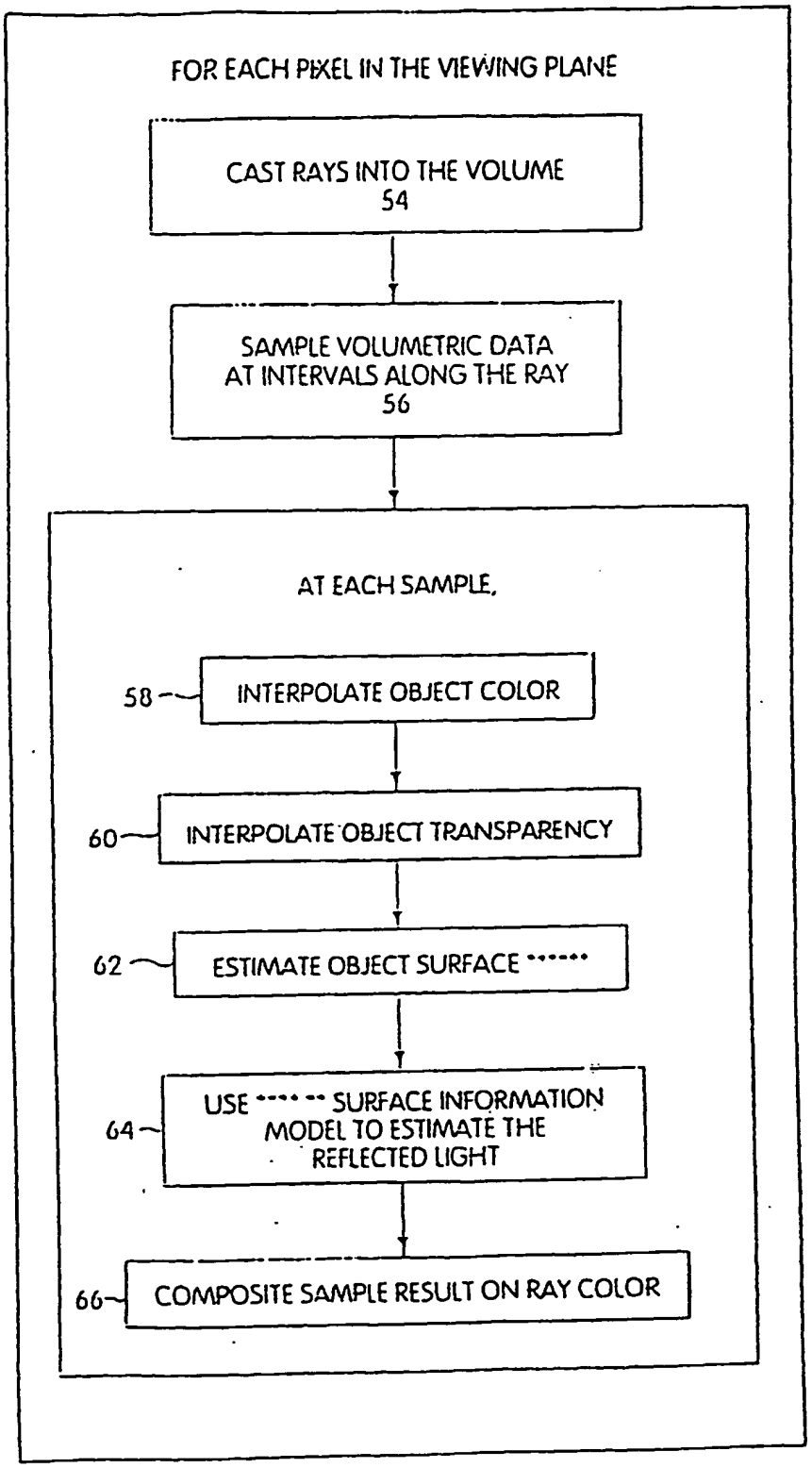
FOR EACH PIXEL IN THE VIEWING PLANE
CAST RAYS INTO THE VOLUME
54
SAMPLE VOLUMETRIC DATA
AT INTERVALS ALONG THE RAY
56
AT EACH SAMPLE,
58
INTERPOLATE OBJECT COLOR
60
INTERPOLATE OBJECT TRANSPARENCY
62
ESTIMATE OBJECT SURFACE
64
USE SURFACE INFORMATION
MODEL TO ESTIMATE THE
REFLECTED LIGHT
66
COMPOSITE SAMPLE RESULT ON RAY COLOR

Fig. 4

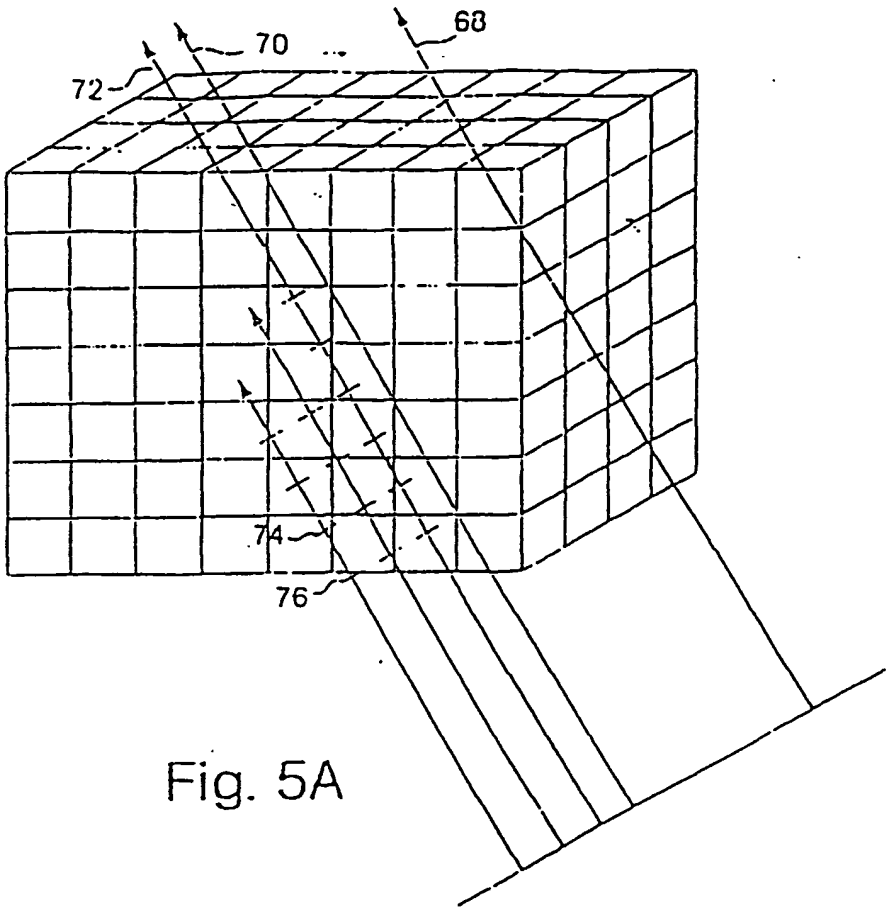

Fig. 5A

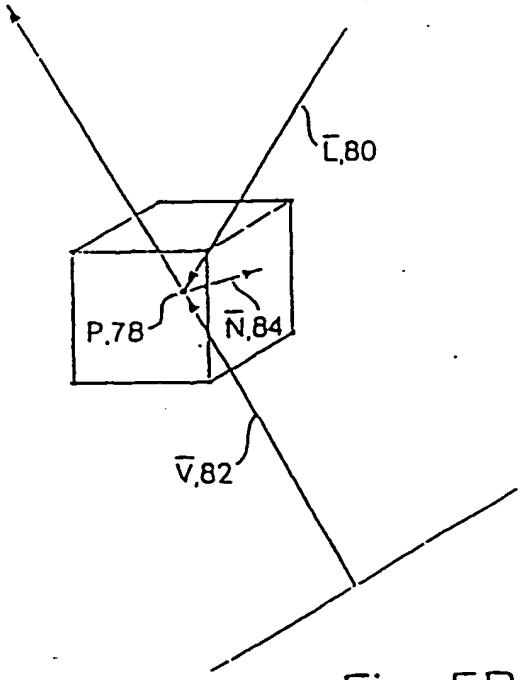

Fig. 5B

```
TYPE DEF VOLUME ELEMENT STRUCT {
86 ——► INT DENSITY;
} VOLUME ELEMENT;
```

Fig. 6A

```
TYPE DEF VOLUME ELEMENT STRUCT {
INT DENSITY;
88 ——► INT DISTANCE;
} VOLUME ELEMENT;
```

Fig. 6B

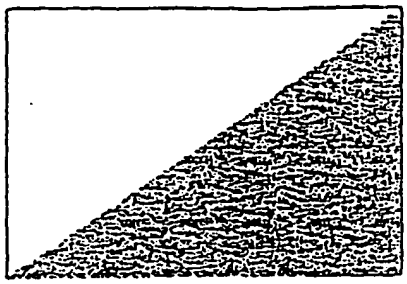

Fig. 7A

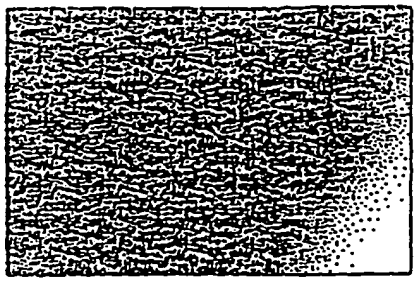

Fig. 7B

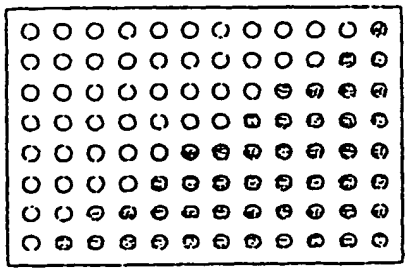

Fig. 7C

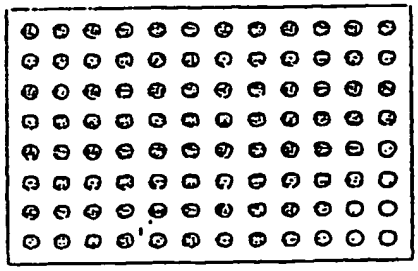

Fig. 7D

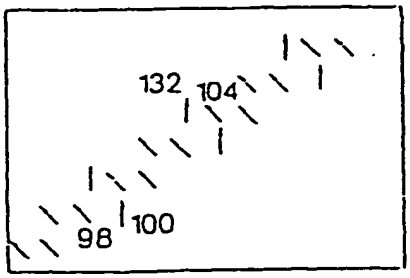

Fig. 7E

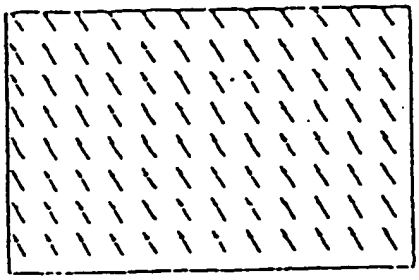

Fig. 7F

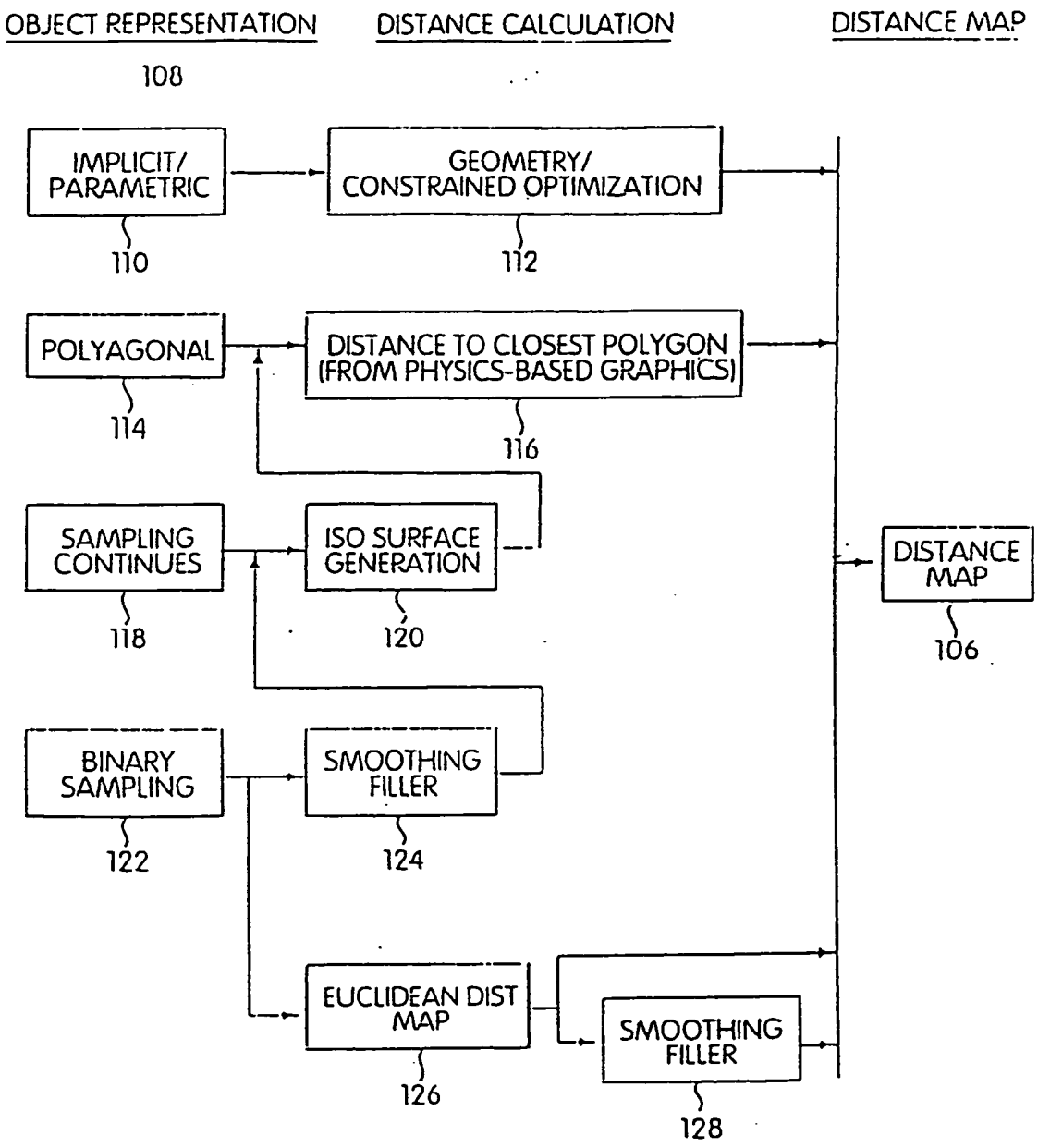
OBJECT REPRESENTATION
DISTANCE CALCULATION
DISTANCE MAP
108
IMPLICIT/ PARAMETRIC
110
GEOMETRY/ CONSTRAINED OPTIMIZATION
112
POLYAGONAL
114
DISTANCE TO CLOSEST POLYGON (FROM PHYSICS-BASED GRAPHICS)
116
SAMPLING CONTINUES
118
ISO SURFACE GENERATION
120
DISTANCE MAP
106
BINARY SAMPLING
122
SMOOTHING FILLER
124
EUCLIDEAN DIST MAP
126
SMOOTHING FILLER
128

Fig. 8